Spring Boot实际应用讲解（六）： MySQL + Spring-data-jpa(Hibernate)


ZYRzyr (/u/f8ff63b17fc7) + 关注
2017.11.27 16:54* 字数 1127 阅读 330 评论 0 喜欢 3
(/u/f8ff63b17fc7)

> 文/ZYRzyr (https://www.jianshu.com/u/f8ff63b17fc7)
> 原文链接:http://www.jianshu.com/p/b204472d8126
> (https://www.jianshu.com/p/b204472d8126)

本文提纲

一、简介
二、实例
三、最后

本文运行环境

```
Ubuntu 16.04 LTS
JDK 8 +
IntelliJ IDEA ULTIMATE 2017.2
Maven 3.5.0
Spring Boot 1.5.8.RELEASE
```

一、简介

相信大家对 MySQL 都很熟悉了，就不多言。
JPA 可以简单的理解为一种保存数据的规范，而 hibernate 是实现这种规范的具体操作。所以本文主要讲解在 Spring Boot 项目中，如何使用 JPA 即 hibernate 将项目中的数据保存到 MySQL 数据库中。

二、实例

2.1 添加依赖

在 pom.xml 中添加如下依赖：

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

2.2 配置文件

在配置文件 application.yml 中添加 MySQL 配置：

```
spring:
  profiles:
    active: pro #使用生成环境配置

  datasource:
      #MySQL数据库驱动
      driver-class-name: com.mysql.jdbc.Driver
      #数据库连接地址
      url: jdbc:mysql://localhost:3306/demo?useSSL=false
      #数据库连接名
      username: root
      #数据库连接密码
      password: 123456
```

在 `application-pro.yml` 中添加 `jpa` 的配置：

```
spring:
  jpa:
    #hibernate实现jpa
    hibernate:
      #数据表的创建方式
      ddl-auto: create-drop
    #是否在控制台显示SQL语句
    show-sql: true
```

`ddl-auto`： 自动创建表的方式，共有5种： `update` 、 `create-drop` 、 `create` 、 `none` 、 `validate` ， 分别表示：

- `update` -创建表，如表结构有变则更新；
- `create-drop` -每次项目启动删除之前表并新建，项目停止时删除所有表，本文选择此项，方便测试；
- `create` -每次项目启动删除之前表并新建；
- `none` -不使用自动创建功能；
- `validate` -验证表结构等，但不对数据库做修改。

此处只配置了最基本的使用，可根据实际情况配置其它需要。

2.3 新建实体类

```
package com.zyr.demo.domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;

@Entity //表示该类是一个实体类，在配置中写了 ddl-auto，jpa会将类名作为表名自动生成表
public class Account {

    @Id   //主键约束
    @GeneratedValue  //自增
    private Integer id;

    @NotNull
    private String name;

    @NotNull
    private String nickName;

    @NotNull
    private Double money;

    public Account() {   //必须写无参构造方法，否则报错
    }

    public Account(Integer id) {
        this.id  = id;
    }

    public Account(String name, String nickName, Double money) {
        this.name = name;
        this.nickName = nickName;
        this.money = money;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getNickName() {
        return nickName;
    }

    public void setNickName(String nickName) {
        this.nickName = nickName;
    }

    public Double getMoney() {
        return money;
    }

    public void setMoney(Double money) {
        this.money = money;
    }

    @Override
    public String toString() {
        return "Account{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", nickName='" + nickName + '\'' +
                ", money=" + money +
                '}';
    }
}
```

1. `@Entity`： 表示该类是一个实体类，在配置中写了 ddl-auto，jpa会将类名作为表名自动生成表;
2. `@Id`： 表示该字段作为主键，等同于 `SQL` 中的 `PRIMARY KEY`；
3. `@GeneratedValue`： 表示 `Integer` 字段自增，等同于 `SQL` 中的 `AUTO INCREMENT`；
4. `@Entity` 标注的类，必须有一个无参构造方法，否则会报错；
5. `@NotNull`： 表示该字段在表中不能为空。

2.4 Service层

通常业务逻辑都放在 `Service` 层中，新建 `AccountService` ，其中包含增删改查，对应 `RESTful API` 中的 `POST` 、 `DELETE` 、 `PUT` 、 `GET`：

```
package com.zyr.demo.service;

import com.zyr.demo.domain.Account;

import java.util.List;

public interface AccountService {
    Account insertAccount(Account account);

    List<Account> findByName(String name);

    void deleteById(Integer id);

    Account updateAccount(Account account);
}
```

接口实现如下：

```
package com.zyr.demo.service.impl;

import com.zyr.demo.domain.Account;
import com.zyr.demo.repository.AccountRepository;
import com.zyr.demo.service.AccountService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Transactional
@Service
public class AccountServiceImpl implements AccountService {

    private AccountRepository accountRepository;

    @Autowired
    public AccountServiceImpl(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

    @Override
    public Account insertAccount(Account account) {
        if (account != null) {
            accountRepository.save(account);
        }
        return account;
    }

    @Override
    public List<Account> findByName(String name) {
        return accountRepository.findByName(name);
    }

    @Override
    public void deleteById(Integer id) {
        accountRepository.delete(new Account(id));
    }

    @Override
    public Account updateAccount(Account account) {
        return accountRepository.save(account);
    }
}
```

1. `@Transactional`： 写在类上表示该类中的所有方法都进行事务管理；
2. `@Service`： 将该类放入 `Spring` 容器中，并表示它是一个 `Service`；

2.5 DAO层

使用 `jpa` 的话， `DAO` 层很“薄”，即内容很简单，放在包 `repository` 下。
新建 `AccountRepository` 接口：

```
package com.zyr.demo.repository;

import com.zyr.demo.domain.Account;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface AccountRepository extends JpaRepository<Account, Integer> {
    List<Account> findByName(String name);
}
```

1. `@Repository`： 将该接口放入 `Spring` 容器中，并表示它是一个 `Repository`；
2. 必须是 `interface`， 并继承 `JpaRepository`；

3. 继承 `JpaRepository` 后，即可直接使用其中已有的方法，如 `save` 、 `findAll` 来保存或查找数据等；
4. 可在该接口中，自定义操作数据库的方法，方法名有一定的规范，新增方法时，输入 `void` 空格，然后 `Ctrl + 空格` 会提示方法名规范。也可完全自定义，即自己写 `HQL` 语句进行操作数据库，网上有很多更好的介绍，本文不过多介绍；

2.6 测试Service

新建测试类 `AccountServiceImplTest`：

```
package com.zyr.demo.service.impl;

import com.zyr.demo.domain.Account;
import com.zyr.demo.service.AccountService;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import java.util.List;

import static org.junit.Assert.*;

@RunWith(SpringRunner.class)
@SpringBootTest
public class AccountServiceImplTest {

    @Autowired
    private AccountService accountService;

    private String name1 = "Bob";
    private String nickName1 = "boooob";
    private Double money1 = 50.5;

    private String name2 = "Tom";
    private String nickName2 = "toooom";
    private Double money2 = 60.5;

    @Before
    public void setUp() throws Exception {
        Account account1 = new Account(name1, nickName1, money1);
        Account account2 = new Account(name2, nickName2, money2);
        accountService.insertAccount(account1);
        accountService.insertAccount(account2);
    }

    @Test
    public void testInsert_and_Find() throws Exception {
        Account account = accountService.findByName(name1).get(0);
        assertEquals(name1, account.getName());
        assertEquals(nickName1, account.getNickName());
        assertEquals(money1, account.getMoney(), .001);
    }

    @Test
    public void testDeleteById() throws Exception {
        accountService.deleteById(1);
        List<Account> accounts = accountService.findByName(name1);
        for (Account account : accounts) {
            assertTrue(account.getId() != 1);
        }
    }

    @Test
    public void testUpdateAccount() throws Exception {
        Account account = new Account("Jack", "jaacck", 200.5);
        account.setId(2);
        accountService.updateAccount(account);
        Account newAccount = accountService.findByName("Jack").get(0);
        assertEquals("Jack", newAccount.getName());
        assertEquals("jaacck", newAccount.getNickName());
        assertEquals(200.5, newAccount.getMoney(), .001);
        assertEquals(2, newAccount.getId(), .001);
    }

}
```

运行测试类，用例全部通过，说明 `service` 层和 `dao` 层没问题；

2.7 新建AccountController

```
package com.zyr.demo.controller;

import com.zyr.demo.domain.Account;
import com.zyr.demo.service.AccountService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/account")
public class AccountController {

    private AccountService accountService;

    @Autowired
    public AccountController(AccountService accountService) {
        this.accountService = accountService;
    }

    @PostMapping("/account")
    public Account insertAccount(Account account) {
        return accountService.insertAccount(account);
    }

    @GetMapping("/accounts")
    public List<Account> findByName(String name) {
        return accountService.findByName(name);
    }

    @PutMapping("/account")
    public Account updateAccount(Account account) {
        return accountService.updateAccount(account);
    }

    @DeleteMapping("/account")
    public void deleteById(Integer id) {
        accountService.deleteById(id);
    }
}
```

2.8 测试AccountController

新建测试类 `AccountControllerTest` 如下：

```
package com.zyr.demo.controller;

import com.zyr.demo.domain.Account;
import com.zyr.demo.service.AccountService;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

import static org.junit.Assert.*;

@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class AccountControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private AccountService accountService;

    private String name = "Bob";
    private String nickName = "boooob";
    private Double money = 50.5;

    @Before
    public void setUp() throws Exception {
        Account account = new Account(name, nickName, money);
        accountService.insertAccount(account);
    }

    @Test
    public void testInsertAccount() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post("/account/account")
                .param("name", "Tom")
                .param("nickName", "tooom")
                .param("money", "10.5"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().json("{\"id\":4,\"name\":
    }

    @Test
    public void testFindByName() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/account/accounts")
                .param("name", name))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().json("[{\"id\":1,\"name\"
    }

    @Test
    public void testUpdateAccount() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.put("/account/account")
                .param("id", "1")
                .param("name", "Tom")
                .param("nickName", "tooom")
                .param("money", "10.5"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().json("{\"id\":1,\"name\":
    }

    @Test
    public void testDeleteById() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.delete("/account/account")
                .param("id", "1"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().string(""));
    }
}
```

运行测试类，用例全部通过，说明 API 无误，同时也可以使用 REST Client 进行测试，这里就不再赘述。

三、最后

本文介绍了在 `Spring Boot` 中如何使用 `JPA`，并演示了其基础用法。文中有不全之处，比如没有 `service` 层或 `Controller` 出错之后的处理、请求有错的处理等等，将在下一篇文章：统一异常处理，全部弥补。

本文代码已上传至我的GitHub仓库 (https://link.jianshu.com?t=https://github.com/ZYRzyr/SpringBootDemo)，进入以后将branches (https://link.jianshu.com?t=https://github.com/ZYRzyr/SpringBootDemo/branches)切换为6-jpa (https://link.jianshu.com?t=https://github.com/ZYRzyr/SpringBootDemo/tree/6-jpa)即可看见。

前篇：
Spring Boot实际应用讲解（一）： Hello World (https://www.jianshu.com/p/60f7e025c680)
Spring Boot实际应用讲解（二）： 配置详解 (https://www.jianshu.com/p/d4c7f33c9b37)
Spring Boot实际应用讲解（三）： 表单验证 (https://www.jianshu.com/p/a2b4e61b5532)
Spring Boot实际应用讲解（四）： RESTful API (https://www.jianshu.com/p/e907595e9d1d)
Spring Boot实际应用讲解（五）： AOP之请求日志 (https://www.jianshu.com/p/93216bf41182)

后续将推出以下文章，敬请关注！

Spring Boot实际应用讲解（七）： 统一异常处理
Spring Boot实际应用讲解（八）： MySQL + Mybatis
Spring Boot实际应用讲解（九）： MySQL + Mybatis + Redis

文中若有错之处，还请各位批评指正，谢谢！

> 原文作者/ZYRzyr (https://www.jianshu.com/u/f8ff63b17fc7)
> 原文链接:http://www.jianshu.com/p/b204472d8126 (https://www.jianshu.com/p/b204472d8126)

ZYRzyr (/u/f8ff63b17fc7) ♂
写了 17755 字，被 32 人关注，获得了 92 个喜欢 (/u/f8ff63b17fc7)

+关注

程序猿一枚 技能树加点情况： 移动端—Android——5/5 移动端—iOS——1/5 前端—JavaScript、HTML—…

喜欢的老铁，来一波关注666

赞赏支持